

Phing

Steffen Wörsdörfer



Inhalt



1. Einleitung
2. Phing
3. Installation
4. Builddatei
5. Basiskomponenten
6. Eingebaute Funktionen
7. Erweiterungen
8. Phing im eStudy Portal
9. Fazit

I. Einleitung



- Was machen Buildwerkzeuge (compiliert) ?
 - fassen die Compileraufrufe zusammen
 - erstellen ein ausführbares Release
- Was machen Buildwerkzeuge (interpretiert) ?
 - ermöglicht die Durchführung verschiedener Abläufe
 - erstellen ein zur Veröffentlichung bereites Release

Im Bereich der PHP Entwicklung wird das Buildwerkzeug **phing** eingesetzt.

2. Phing

Grundlegendes:

- basiert auf Apache Ant
- geschrieben in PHP
- erweiterbar durch PHP
- benutzt XML als Builddatei - Sprache
- Funktionalität wie andere Buildprogramme

Phing is not GNU make



2. Phing

Historisches:

- entstand aus dem PHP Framework BinaryCloud
- erstellt dort PHP Arrays aus XML Metadaten
- ersetzt, wegen der Abhängigkeit von Unix, das Buildsystem GNU Make
- als Vorbild dient das Apache Ant Projekt
- unterstützt seit Phing 2 auch PHP 5



3. Installation



Voraussetzungen (Phing 2.2.0) :

- Windows oder Unix ähnliches Betriebssystem
- CLI Version von min. PHP 5.0.1
- Webserver mit XML und XSL

3. Installation



Installations - Variante 1 (pear) :

- automatische Installation mittels pear installer

Installations - Variante 2 (manuell) :

- entpacken der pear Pakete
- Anpassen und Setzen von Umgebungsvariablen

4. Builddatei



XML Builddatei:

- fordert wohlgeformtes XML
- fordert XML Dokumenten Header:

```
<?xml version="1.0"?>
```

- fordert genau ein Wurzelement:

```
<project name="proName" default="defTarget">  
  <!-- ... -->  
</project>
```

- fordert min. ein Target:

```
<target name="defTarget">  
  <!-- ... -->  
</target>
```


4. Builddatei



Daraus folgt als Minimum:

```
<?xml version="1.0"?>
<project name="proName" default="defTarget">

    <target name="defTarget">
        <!-- hier stehen die Targetspezifischen Tasks -->
    </target>

</project>
```

4. Builddatei



Aufrufe haben folgende Struktur:

phing [options] [target [target2 [target3] ...]]

Beispiele:

```
phing
```

```
phing -f msg.xml
```

```
phing -f msg.xml output2
```

```
phing -f msg.xml -Dmessage2=test output2
```


4. Builddatei



Beispiel Nr. 1

5. Basiskomponenten



Unterschieden werden 5 Basiskomponenten:

- project
- target
- task
- type
- property

5. Basiskomponenten



project:

- Wurzelement eines jeden Projekts
- muss genau einmal vorkommen

Beispiel:

```
<project name="projectName"
         default="defTarget"
         basedir=". ">
  <!-- ... -->
</project>
```

5. Basiskomponenten



target:

- Sammlung von Komponenten
- kann von einem anderem targets abhängig sein

Beispiel:

```
<target name="targetName"  
    depends="targetDependsOn"  
    description="your target description">  
    <!-- ... -->  
</target>
```


5. Basiskomponenten



task:

- definiert die eigentliche Arbeit
- können einfach oder verschachtelt sein

Beispiel:

```
<echo>Building...</echo>  
<copy todir="./build">  
  <!-- ... -->  
</copy>
```

5. Basiskomponenten



type:

- kann einfacher einfach (string, int, ...) oder komplex (verschachtelt) sein
- kann referenziert werden

Beispiel:

Definition:

```
<fileset dir="./WebDevPHP" id="srcFiles">  
  <include name="**/*" />  
</fileset>
```

Anwendung:

```
<fileset refid="srcFiles"/>
```


5. Basiskomponenten



property:

- ähneln Variablen
- es gibt eingebaute (built-in) property:
 - phing.file (vollständiger Pfad zur aktuellen Builddatei)
 - phing.project.name (Name des aktuellen Projekts)
 - projet.basedir (Basisverzeichnis des aktuellen Projekts)
 - usw.

Beispiel:

Definition:

```
<property name="propName" value="propValue123"/>
```

Anwendung:

```
<echo msg="${propName}" />  
<echo msg="${phing.file}" />
```

5. Basiskomponenten



Beispiel Nr. 2

6. Eingebaute Funktionen



Es wird zwischen folgendem unterschieden:

- core tasks
- optional tasks
- core types

6. Eingebaute Funktionen



core tasks:

- arbeiten ohne Mithilfe externer Tools

Beispiele:

- diverse Dateisystem Tasks (copy, delete, ...)
- Kontrollstrukturen (condition, foreach, if, ...)

6. Eingebaute Funktionen



optional tasks:

- arbeiten nur mit Hilfe externer Tools

Beispiele:

- PHPUnit
- CodeCoverage Report
- PHPDocumentor
- SVN
- Tar, Zip

6. Eingebaute Funktionen



core types:

- es ist eine solide Basis an types vorhanden

Beispiele:

- Filelist, Fileset
- diverse Filter
- diverse Mapper

7. Erweiterungen



Erweiterungen können in 3 Bereichen erfolgen:

- task
- type
- mapper

7. Erweiterungen



task:

- häufigste Art der Programmierung von Erweiterungen

type:

- wird selten genutzt
- sollte nur bei Mehrfachbenutzung erstellt werden

mapper:

- wird selten genutzt
- meistens wird RegExpMapper oder GlobMapper verwendet

7. Erweiterungen



task:

- Speicherung in *phing/tasks/*
- Bekanntmachung durch

```
<taskdef name="myTask" classname="phing.tasks.MyTask" />
```

type:

- Speicherung in *phing/types/*
- Bekanntmachung durch

```
<taskdef name="myType" classname="phing.types.MyType" />
```

mapper:

- Speicherung in *phing/mappers/*
- Bekanntmachung durch

```
<mapper classname="myapp.mappers.MyMapper" />
```

7. Erweiterungen



Vorgaben:

- einhalten der PEAR Coding Standards
- Dateinamen bestehen aus genau 2 Elementen: Dateiname + Typkürzel
- Kurze beschreibende Dateinamen
- Namen dürfen keine Punkte enthalten
- Dateien mit PHP Code müssen auf .php enden
- Builddateien müssen mit .xml enden
- Es muss genau eine Klasse pro Datei bestehen
- Der Namensteil der Datei muss genau dem Klassennamen der beinhalteten Klasse entsprechen

7. Erweiterungen



Allgemeines (Pakete importieren):

- jeweiligen Pakete anhand des Einsatzziels importieren

task:

- `require_once "phing/Task.php";`

type:

- `require_once "phing/types/DataType.php";`

mapper:

- `require_once "phing/mappers/FileNameMapper.php";`

7. Erweiterungen



Allgemeines (Klassen Deklaration und Definition):

- jede neue Klasse implementiert oder erweitert eine Basisklasse oder Schnittstelle

task:

- ```
class MyTask extends Task {
 ...
}
```

### type:

- ```
class MyType extends DataType {  
    ...  
}
```

mapper:

- ```
class MyMapper implements FileNameMapper {
 ...
}
```



# 7. Erweiterungen



## Allgemeines (Klasseneigenschaften):

- in jeder neuen Klasse hat man die Möglichkeit Werte in Klasseneigenschaften zu speichern

### task:

- Vorsicht, viele Eigenschaften werden von der Superklasse geerbt

### type:

- sind ein sehr wichtiger Teil bei der type Programmierung, da es sich ja um einen Daten- bzw. Wertespeicher handelt.

### mapper:

- dient zum Speichern der setFrom und setTo Werte

# 7. Erweiterungen



Speziell (**task**):

Klassen Konstruktor:

- initialisiert die Klasseneigenschaften

Setter Methoden für XML Attribute:

- Methoden sind public und Methodename beginnt mit set
- müssen für jedes Attribut programmiert werden

Creator Methoden:

- verwalten geschachtelte XML Tags
- Methodename beginnt immer mit create

main() Methode:

- Eintrittspunkt der Klasse, welche den Ablauf bestimmt

init() Methode:

- überprüft beim Parsen der XML Datei auf erfolgreiche Voraussetzungen



# 7. Erweiterungen



Speziell (**type**):

Klassen Konstruktor:

- initialisiert die Klasseneigenschaften

Setter Methoden für XML Attribute:

- Methoden sind public und Methodename beginnt mit set
- müssen für jedes Attribut programmiert werden

Getter Methoden für XML Attribute:

- Methoden sind public und Methodename beginnt mit get
- müssen für jedes Attribut programmiert werden

aktuelles Objekt überprüfen:

- überprüft auf Typ Korrektheit

# 7. Erweiterungen



Speziell (**mapper**):

main(\$sourceFileName) Methode:

- Eintrittspunkt der Klasse
- verarbeitet übergebene Datei und gibt sie angepasst zurück

setFrom(\$from) Methode:

- muss implementiert jedoch nicht notwendigerweise definiert werden

setTo(\$to) Methode:

- muss implementiert jedoch nicht notwendigerweise definiert werden



# 7. Erweiterungen



## Beispiel Nr. 3

# 8. Phing im eStudy Portal



## Mögliche Einsatzgebiete:

- überprüfen der Module mit PHPUnit
- erstellen der Dokumentation mit PHPDocumentor
- überprüfen der PHP Dateien mittels PHPLint
- administrative Aufgaben



# 9. Fazit



## Pro:

- einfache Erstellung von Webreleases
- syntaktische Überprüfung von Quelldateien
- Zeitersparnis für immer wiederkehrende Aufgaben

## Contra:

- momentan noch fehlerhafte Dokumentation
- Inkompatibilität mit PHPUnit 3.0.2



Vielen Dank für Ihre Aufmerksamkeit

# Diskussion